Novel RNS Parameter Selection for Fast Modular Multiplication

Gavin Xiaoxu Yao, *Student Member, IEEE,* Ray C.C. Cheung, *Member, IEEE,* Junfeng Fan, *Student Member, IEEE,* and Ingrid Verbauwhede, *Fellow, IEEE*

Abstract—The parameter selection of Residue Number Systems (RNS) has a great impact on its computational efficiency. This paper shows that a base extension, the most costly operation in RNS Montgomery multiplication, can be more efficient when the intervals between the RNS moduli are small. We propose a systematic RNS parameter selection procedure and two methods to select RNS moduli that lead to a reduced complexity. Our experimental results confirm the advantages of the selected moduli.

Index Terms—Modular multiplication, Residue Number System (RNS), base extension

1 Introduction

Modular multiplication of long integers is a basic operation in Public-Key Cryptography (PKC) [1]. The size of the operands ranges from a few hundred bits (e.g. 256-bit Elliptic Curve Cryptography, ECC [2], [3]) to several hundred thousand bits (e.g. 741455-bit fully homomorphic encryption [4]). Building a high-throughput long-integer modular multiplier is thus the key step towards a high-performance PKC implementation.

Parallelization has been one of the most powerful methods to accelerate arithmetic operations in both hardware and software implementations [5]. The operations in a Residue Number System (RNS) are distributed into a group of small integers, and are naturally suitable for parallel implementations. Thus, the Montgomery modular algorithms [6] were proposed in the RNS context for cryptographic applications [7], [8], and the parallelism provides the capability of high speed modular multiplication [9], [10], [11], [12].

However, the complexity of the RNS Montgomery is slightly higher than the conventional Montgomery. Digit-serial Montgomery algorithm [13] only uses $2n^2+n$ $w\times w$-bit multiplications, while RNS version requires $2n^2+4n$ [14] for computing $AB \bmod P$, where A, B and P have n w-bit digits. The most computationally expensive step of the RNS Montgomery algorithm is the Base Extension (BE). Out of the $2n^2+4n$ digit multiplications, $2n^2+2n$ multiplications are due to the BEs.

- *G. Yao and R. Cheung are with the Department of Electronic Engineering, City University of Hong Kong, Hong Kong SAR, China E-mail: gavin.yao@student.cityu.edu.hk, r.cheung@cityu.edu.hk*
- *J. Fan and I. Verbauwhede are with the Department of Electrical Engineering, KU Leuven and IBBT, ESAT/SCD-COSIC, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium. E-mail: {jfan,iverbauw}@esat.kuleuven.be J. Fan is the corresponding author.*

In this paper, we first show that if the difference between any two moduli of the RNS bases are small, i.e. all the moduli are *close*, the operand size of the $2n^2$ multiplications in BEs can be reduced from $w\times w$-bit to $v\times w$-bit, where v could be much smaller than w. Consequently, the $2n^2+4n$ $w\times w$-bit multiplications can be substituted by $2n^2$ $v\times w$-bit and $5n$ $w\times w$-bit multiplications. Two methods are provided to search for such kind of close moduli, and a systematic RNS parameter selection procedure is described to achieve the minimum complexity.

The rest of the paper is organized as follows. Section 2 gives a recap of the mathematical background. Section 3 describes the specifications of bases to achieve faster base extensions. The complexity of the refined RNS algorithm is analyzed in Section 4. We provide two methods to select RNS moduli in Section 5. Section 6 discusses briefly the related work and the implementation results. We conclude this work in Section 7.

Due to the nature of this subject, we need a great amount of variables. For the ease of reading, we summarize all the notations in Table 1. The definition is given when appropriate in the context.

2 Background

In this section, we give a brief introduction to RNS arithmetic and RNS Montgomery algorithm.

2.1 Residue Number System

A Residue Number System represents a large integer with several smaller integers. An RNS is defined by a set of n pairwise coprime integer constants:

$$\mathfrak{B} = \{b_1, b_2, \ldots, b_n\}.$$

The set $\mathfrak{B}$ is also known as a *base*, and the element b_i, $1 \leqslant i \leqslant n$, is called an *RNS modulus*, and each modulus forms an RNS *channel*. Let $M_{\mathfrak{B}} = \prod_{i=1}^{n} b_i$. Let $|a|_b$ be a modulo b, then any integer X, $0 \leqslant X < M_{\mathfrak{B}}$, can

TABLE 1: Notations used in RNS Montgomery Algorithm

Symbol	Definition
$\|x\|_y$	$x \bmod y$
P	The modulus
w	The bit-length of RNS moduli
$\mathfrak{B}$	The first RNS base: $\{b_1, b_2, \ldots, b_n\}$
$\mathfrak{C}$	The second RNS base: $\{c_1, c_2, \ldots, c_n\}$
n	The number of RNS moduli of $\mathfrak{B}$ or $\mathfrak{C}$*
B_i	The constant defined by $\mathfrak{B}$, $B_i = \prod_{j=1, j\neq i}^{n} b_j$
C_i	The constant defined by $\mathfrak{C}$, $C_i = \prod_{j=1, j\neq i}^{n} c_j$
B_i^{-1}	$\|B_i^{-1}\|_{b_i}$
C_i^{-1}	$\|C_i^{-1}\|_{c_i}$
$\tilde{B}_{i,j}$	The constant to substitute $\|B_i\|_{c_j}$
$\tilde{B'}_{i,j}$	$\tilde{B}_{i,j}$ after zero truncation
d_i	$d_i = 2^w - b_i$, $d_i < 2^{\lfloor \frac{w}{2} \rfloor}$
$M_{\mathfrak{B}}$	The dynamic range defined by $\mathfrak{B}$, $M_{\mathfrak{B}} = \prod_{i=1}^{n} b_i$
$M_{\mathfrak{C}}$	The dynamic range defined by $\mathfrak{C}$, $M_{\mathfrak{C}} = \prod_{i=1}^{n} c_i$
m	The bit-length of a machine word
s	The number of machine words for a w-bit integer
t	The number of machine words for a v-bit integer
v	The maximum bit-length of all $\tilde{B}_{i,j}$
v'	The maximum bit-length of all $\tilde{B'}_{i,j}$
q_i	The i-th element of $\{Q\}_{\mathfrak{B}}$
q'_j	The j-th element of $\{Q\}_{\mathfrak{C}}$
λ	An intermediate value in BE, c.f. (4)
$\hat{\lambda}$	Estimation of λ
$\xi_{Q,i}$	The i-th element of an intermediate value in BE of Q, $\xi_{Q,i} = \|q_i \cdot B_i^{-1}\|_{b_i}$

* We choose $nw \geqslant \lceil \log_2 3P \rceil$. If lazy reduction is used, the dynamic range should be enlarged correspondingly.

be uniquely represented by a set of smaller integers:

$$\{X\}_{\mathfrak{B}} = \{|X|_{b_1}, |X|_{b_2}, \ldots, |X|_{b_n}\}.$$

We denote w the bit-length of b_i.

The original value of X can be restored from $\{X\}_{\mathfrak{B}}$ using the Chinese Remainder Theorem (CRT):

$$X = \left| \sum_{i=1}^{n} \xi_i \cdot B_i \right|_{M_{\mathfrak{B}}} := \left| \sum_{i=1}^{n} |x_i \cdot B_i^{-1}|_{b_i} \cdot B_i \right|_{M_{\mathfrak{B}}}, \quad (1)$$

$$B_i = M_{\mathfrak{B}}/b_i = \prod_{j=1, j\neq i}^{n} b_j, \ 1 \leqslant i \leqslant n. \quad (2)$$

Using RNS, arithmetic operations in $\mathbb{Z}/M_{\mathfrak{B}}\mathbb{Z}$ can be efficiently performed. Consider two integers X, Y and their RNS representations $\{X\}_{\mathfrak{B}} = \{x_1, x_2, \ldots, x_n\}$ and $\{Y\}_{\mathfrak{B}} = \{y_1, y_2, \ldots, y_n\}$, then

$$\{|X \odot Y|_{M_{\mathfrak{B}}}\}_{\mathfrak{B}} = \{|x_1 \odot y_1|_{b_1}, \ldots, |x_n \odot y_n|_{b_n}\}.$$

for $\odot \in \{+, -, \times, /\}$. The division is available only if the multiplicative inverse of Y exists in $\mathfrak{B}$, i.e. Y is coprime with $M_{\mathfrak{B}}$.

For every operation, there is a channel reduction followed to reduce the result to the range $[0, b_i)$. In order to accelerate the channel reduction, pseudo-Mersenne numbers of the form $b_i = 2^w - d_i$, where $d_i < 2^{\lfloor \frac{w}{2} \rfloor}$, are selected as moduli. The channel reduction modulo b_i can then be performed by Algorithm 1. If the Hamming weight of d_i is small, multiplications by d_i can also be replaced by a few additions.

The RNS is appealing because it is naturally suitable for parallel implementations. For all the basic operations $(+, -, \times, /)$, computations between x_i and y_i have no dependency on other channels, which largely simplifies the parallelization of the operations. Fig. 1 shows the top level structure of an RNS multiplier. Note that each channel uses only local memory once the data is loaded, which also simplifies the communication and the scheduling.

Algorithm 1 Reduction modulo $(2^w - d_i)$

Require: $x < 2^{2w}$, w, $0 < d_i < 2^{\lfloor w/2 \rfloor}$
Ensure: $r \equiv x \bmod (2^w - d_i)$, $r < 2^w$
1: $r \leftarrow (x \bmod 2^w) + d_i \cdot \lfloor x/2^w \rfloor$
2: $r \leftarrow (r \bmod 2^w) + d_i \cdot \lfloor r/2^w \rfloor$
3: **if** $r > 2^w$ **then**
4: $\quad r \leftarrow (r \bmod 2^w) + d_i$
5: **end if**
6: **return** r

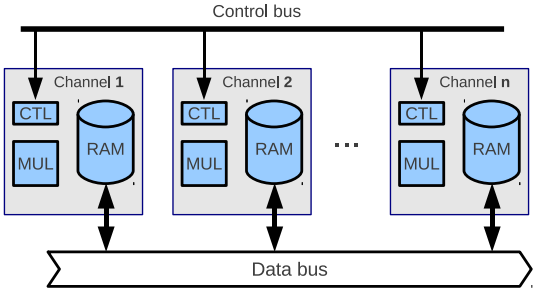


Fig. 1: Parallel implementation of RNS modular multiplication.

Unfortunately, RNS multiplications are always modulo a composite number $M_{\mathfrak{B}}$, and it can hardly be used immediately in cryptographic algorithms. For instance, ECC over $GF(p)$ uses multiplications modulo a prime number. RSA implementations can make use of RNS when the factors of N are known. In order to work with the modulus which is prime or infeasible to factorize, one can combine the RNS representation with the Montgomery reduction algorithm [7], [8].

2.2 RNS Montgomery Algorithm

Algorithm 2 shows the Montgomery modular multiplication algorithm in RNS representation context. A new base, $\mathfrak{C} = \{c_1, c_2, \ldots, c_n\}$, where $M_{\mathfrak{C}} = \prod_{i=1}^{n} c_i$ is coprime with $M_{\mathfrak{B}}$, is introduced to perform the division. Note that all the moduli from both $\mathfrak{B}$ and $\mathfrak{C}$ are pairwise coprime as $M_{\mathfrak{B}}$ and $M_{\mathfrak{C}}$ are coprime. The overhead is two Base Extensions (BEs).

Base Extension To compute $\{Q\}_{\mathfrak{C}} = \{q'_1, q'_2, \ldots, q'_n\}$ from $\{Q\}_{\mathfrak{B}} = \{q_1, q_2, \ldots, q_n\}$, there are two main streams in literature: using CRT [7] and using Mixed Radix number System (MRS) [8]. Since the MRS method is difficult to implement in a full-parallel fashion (c.f. Section 6), we choose the CRT method [15]. Given $\{X\}_{\mathfrak{B}}$, for (1), there must exist an integer $\lambda < n$

Algorithm 2 RNS Montgomery modular multiplication [9]

Require: RNS bases $\mathfrak{B}$ and $\mathfrak{C}$ with $M_{\mathfrak{B}}, M_{\mathfrak{C}} > 2P$
Require: P, $M_{\mathfrak{B}}$, $M_{\mathfrak{C}}$ are pairwise coprime
Require: $\{\bar{X}\}_{\mathfrak{B}}, \{\bar{X}\}_{\mathfrak{C}}$ s.t. $\bar{X} \equiv XM_{\mathfrak{B}} \bmod P$ and $\bar{X} < 2P$
Require: $\{\bar{Y}\}_{\mathfrak{B}}, \{\bar{Y}\}_{\mathfrak{C}}$ s.t. $\bar{Y} \equiv YM_{\mathfrak{B}} \bmod P$ and $\bar{Y} < 2P$
Precompute: $\{P'\}_{\mathfrak{B}} \leftarrow \{|-P^{-1}|_{M_{\mathfrak{B}}}\}_{\mathfrak{B}}$
Precompute: $\{M'\}_{\mathfrak{C}} \leftarrow \{|M_{\mathfrak{B}}^{-1}|_{M_{\mathfrak{C}}}\}_{\mathfrak{C}}$ and $\{P\}_{\mathfrak{C}}$
Ensure: $\{U\}_{\mathfrak{B}}$, $\{U\}_{\mathfrak{C}}$ s.t. $U \equiv XYM_{\mathfrak{B}} \bmod P$, $U < 2P$

	in $\mathfrak{B}$	in $\mathfrak{C}$
1:	$\{T\}_{\mathfrak{B}} \leftarrow \{\bar{X}\}_{\mathfrak{B}} \times \{\bar{Y}\}_{\mathfrak{B}}$,	$\{T\}_{\mathfrak{C}} \leftarrow \{\bar{X}\}_{\mathfrak{C}} \times \{\bar{Y}\}_{\mathfrak{C}}$
2:	$\{Q\}_{\mathfrak{B}} \leftarrow \{T\}_{\mathfrak{B}} \times \{P'\}_{\mathfrak{B}}$	
3:	$\{Q\}_{\mathfrak{B}} \xrightarrow{Base\ Extension\ 1} \{Q\}_{\mathfrak{C}}$	
4:		$\{U\}_{\mathfrak{C}} \leftarrow (\{T\}_{\mathfrak{C}} + \{Q\}_{\mathfrak{C}} \times \{P\}_{\mathfrak{C}}) \times \{M'\}_{\mathfrak{C}}$
5:	$\{U\}_{\mathfrak{B}} \xleftarrow{Base\ Extension\ 2} \{U\}_{\mathfrak{C}}$	
6:	**return** $\{U\}_{\mathfrak{B}}$ and $\{U\}_{\mathfrak{C}}$	

such that:

$$Q = \left| \sum_{i=1}^{n} \xi_{Q,i} \cdot B_i \right|_{M_{\mathfrak{B}}} = \sum_{i=1}^{n} \xi_{Q,i} \cdot B_i - \lambda \cdot M_{\mathfrak{B}} \quad (3)$$

where λ can be calculated by the following equation:

$$\lambda = \left\lfloor \sum_{i=1}^{n} \frac{\xi_{Q,i} \cdot B_i}{M_{\mathfrak{B}}} \right\rfloor = \left\lfloor \sum_{i=1}^{n} \frac{\xi_{Q,i}}{b_i} \right\rfloor \quad (4)$$

In [9], $\xi_{Q,i}/b_i$ is further approximated by $\xi_{Q,i}/2^w$ with error correction as b_i is of the form $2^w - d_i, d_i > 0$. Once λ is obtained, $\{Q\}_{\mathfrak{C}} = \{q'_1, \ldots, q'_n\}$ can be computed as follows:

$$q'_j = |Q|_{c_j} = \left| \sum_{i=1}^{n} \xi_{Q,i} \cdot |B_i|_{c_j} - \lambda \cdot |M_{\mathfrak{B}}|_{c_j} \right|_{c_j} \quad (5)$$

The $|B_i|_{c_j}$ and $|M_{\mathfrak{B}}|_{c_j}$, $1 \leqslant i, j \leqslant n$, can be precomputed once $\mathfrak{B}$ and $\mathfrak{C}$ are fixed.

Clearly, we need n $w \times w$-bit multiplications to calculate $\xi_{Q,i}$ and n^2 $w \times w$-bit multiplications for all $\sum_{i=1}^{n} \xi_{Q,i} \cdot |B_i|_{c_j}, 1 \leqslant j \leqslant n$. Therefore, each base extension uses $n^2 + n$ $w \times w$-bit multiplications.[1] In total, one modular multiplication takes $2n^2 + 4n$ $w \times w$-bit multiplications with precomputations [14].

3 Base Selection Specification

In this section, we provide the base selection specification to ensure faster BE operation.

3.1 The Observation on BE Operations

In our previous work [12], [16], we observe that the complexity of BE can be reduced if the moduli in the two bases are close to each other. The BE step, (5), can be written as a matrix-by-vector multiplication followed by channel reductions.

$$\left(\ q''_1, \ldots, q''_n \ \right)^T :=$$

$$\begin{pmatrix} |B_1|_{c_1} & \cdots & |B_n|_{c_1} \\ \vdots & \ddots & \vdots \\ |B_1|_{c_n} & \cdots & |B_n|_{c_n} \end{pmatrix} \begin{pmatrix} \xi_{Q,1} \\ \vdots \\ \xi_{Q,n} \end{pmatrix} - \lambda \begin{pmatrix} |M_{\mathfrak{B}}|_{c_1} \\ \vdots \\ |M_{\mathfrak{B}}|_{c_n} \end{pmatrix} \quad (6)$$

$$\{Q\}_{\mathfrak{C}} \equiv \{q'_1, \ldots, q'_n\} := \{|q''_1|_{c_1}, \ldots, |q''_n|_{c_n}\} \quad (7)$$

Note that the elements in the matrix, $|B_i|_{c_j}$, $1 \leqslant i, j \leqslant n$, are constants and are generated as follows:

$$|B_i|_{c_j} = \left| \prod_{k=1, k \neq i}^{n} b_k \right|_{c_j} = \left| \prod_{k=1, k \neq i}^{n} (b_k - c_j) \right|_{c_j} \quad (8)$$

Define $\tilde{B}_{i,j} := \prod_{k=1, k \neq i}^{n} (b_k - c_j)$. When b_k and c_j are close to each other, the difference $b_k - c_j$ is small. Therefore, the bit-length of $\tilde{B}_{i,j}$ can be much shorter than that of c_j if n is relatively small. Moreover, using $|B_i|_{c_j}$ or $\tilde{B}_{i,j}$ makes no difference in the final results due to the channel reduction on the products. We denote by v the maximal bit-length of $\tilde{B}_{i,j}$. Now the $w \times w$ multiplications are substituted by $v \times w$ multiplications, where $v < w$. [2]

Further Bit-length Reduction If all the $2n$ moduli are odd, $\tilde{B}_{i,j}$ for $1 \leqslant i, j \leqslant n$ will predictably have $(n-1)$ zeros in the Least Significant Bits (LSBs). Since b_k, c_j are all odd, $(b_k - c_j)$ can be divided by 2, and $\tilde{B}_{i,j}$ is then divisible by 2^{n-1}. In practice, $\tilde{B}_{i,j}$ usually has more than $n-1$ zero bits in LSBs.

To further reduce the operand size, we can truncate the least significant zeros of $\tilde{B}_{i,j}$, and restore the correct multiplication results by a simple left-shift. We denote $\tilde{B}'_{i,j}$ for the $\tilde{B}_{i,j}$ after truncation, and v' for the maximal bit-length of $\tilde{B}'_{i,j}$. [3]

3.2 Base Selection Specification

A *good* base thus should satisfy:

- The maximal bit-length of $\tilde{B}_{i,j}$ or $\tilde{B}'_{i,j}$ (i.e. v or v') is minimized;
- The channel reduction is efficient. For instance, b_i is a power of two or a pseudo-Mersenne number.

Previous practices, e.g. [9], [10], [11], tend to consider solely the second condition. The choice of n, the number of moduli in a base, is normally determined after w, the size of integer multipliers in use. For example, it seems natural to choose $w = 18$ or $w = 34$ for FPGA designs since modern FPGAs have 18-bit or 34-bit dedicated multipliers. In the following sections, we will show that this might not be the best choice.

4 Complexity Analysis

We first show all the operations involved in the RNS Montgomery algorithm in Algorithm 3. The algorithm is mainly borrowed from [14] which employ precomputations to reduce the steps, and we modify it to make it suitable to utilize the v-bit operands.

1. $\lambda \cdot |M_{\mathfrak{B}}|_{c_i}$ is negligible since λ is only $\lceil \log_2 n \rceil$ bits.

2. The similar method can also apply to $|M_{\mathfrak{B}}|_{c_j}$.

3. The mechanism of notation also works for the BE from base $\mathfrak{C}$ to base $\mathfrak{B}$, i.e. the second BE in Algorithm 2.

Algorithm 3 RNS Montgomery modular multiplication with pre-computations

Require: RNS bases $\mathfrak{B}$ and $\mathfrak{C}$ with $M_{\mathfrak{B}}, M_{\mathfrak{C}} > 3P$, and ϵ determined by $\mathfrak{B}$ and $\mathfrak{C}$ [9].
Require: $\{\bar{X}\}_{\mathfrak{B}}$ s.t. $\bar{X} \equiv XM_{\mathfrak{B}} \bmod P, \bar{X} < 3P$
Require: $\{\bar{X}'\}_{\mathfrak{C}} = \{\bar{X}\}_{\mathfrak{C}} \times \{C_i^{-1}\}_{\mathfrak{C}}$
Require: $\{\bar{Y}\}_{\mathfrak{B}}$ s.t. $\bar{Y} \equiv YM_{\mathfrak{B}} \bmod P, \bar{Y} < 3P$
Require: $\{\bar{Y}'\}_{\mathfrak{C}} = \{\bar{Y}\}_{\mathfrak{C}} \times \{C_i^{-1}\}_{\mathfrak{C}}$
Precompute: $\{B_i^{-1}P'\}_{\mathfrak{B}} \leftarrow \{|-P^{-1}|_{M_{\mathfrak{B}}}\}_{\mathfrak{B}} \times \{B_i^{-1}\}_{\mathfrak{B}}$
Precompute: $\{C_iM'\}_{\mathfrak{C}} \leftarrow \{|M_{\mathfrak{B}}^{-1}|_{M_{\mathfrak{C}}}\}_{\mathfrak{C}} \times \{C_i\}_{\mathfrak{C}}$
Precompute: $\{PM'C_i^{-1}\}_{\mathfrak{C}} \leftarrow \{P\}_{\mathfrak{C}} \times \{|M_{\mathfrak{B}}^{-1}|_{M_{\mathfrak{C}}}\}_{\mathfrak{C}} \times \{C_i^{-1}\}_{\mathfrak{C}}$
Ensure: $\{U\}_{\mathfrak{B}}$ s.t. $U \equiv XYM_{\mathfrak{B}} \bmod P, U < 3P$
Ensure: $\{U'\}_{\mathfrak{C}} = \{U\}_{\mathfrak{C}} \times \{C_i^{-1}\}_{\mathfrak{C}}$
1: $\{T\}_{\mathfrak{B}} \leftarrow \{X\}_{\mathfrak{B}} \times \{Y\}_{\mathfrak{B}}$
2: $\{T'\}_{\mathfrak{C}} \leftarrow \{X'\}_{\mathfrak{C}} \times \{Y'\}_{\mathfrak{C}}$
3: $\{\xi_Q\}_{\mathfrak{B}} \leftarrow \{T\}_{\mathfrak{B}} \times \{B_i^{-1}P'\}_{\mathfrak{B}}$
4: $\{T''\}_{\mathfrak{C}} \leftarrow \{T'\}_{\mathfrak{C}} \times \{C_iM'\}_{\mathfrak{C}}$
5: $\hat{\lambda} \leftarrow \left\lfloor \sum_{i=1}^{n} \xi_{Q,i}/2^w \right\rfloor$
6: $|Q|_{c_j} \leftarrow \left| \sum_{i=1}^{n} \tilde{B}_{i,j} \cdot \xi_{Q,i} - \hat{\lambda} \cdot \tilde{M}_{\mathfrak{B},j} \right|_{c_j}, 1 \leqslant j \leqslant n$
7: $\{U'\}_{\mathfrak{C}} \leftarrow \{T''\}_{\mathfrak{C}} + \{Q\}_{\mathfrak{C}} \times \{PM'C_i^{-1}\}_{\mathfrak{C}}$
8: $\hat{\lambda} \leftarrow \epsilon + \left\lfloor \sum_{i=1}^{n} U_i'/2^w \right\rfloor$
9: $|U|_{b_j} \leftarrow \left| \sum_{i=1}^{n} \tilde{C}_{i,j} \cdot U_i' - \hat{\lambda} \cdot \tilde{M}_{\mathfrak{C},j} \right|_{b_j}, 1 \leqslant j \leqslant n$
10: **return** $\{U\}_{\mathfrak{B}}$ and $\{U'\}_{\mathfrak{C}}$

Table 2 provides all the operations involved and their executed times in Algorithm 3. The most expensive operations are the $5n$ $w \times w$-bit multiplications in Steps 1–4 and 7, and the $2n^2$ $v \times w$-bit (or $v' \times w$-bit) multiplications in Steps 6 and 9. We also consider the overhead of lazy reduction. Consider $\sum A_iB_i \bmod P$, the complexity increases by $2n$ $w \times w$-bit multiplications for every additional term. For conventional Montgomery, n^2 multiplications are added.

Furthermore, Table 2 provides the number of m-bit multiplications, where m refers to the bit-length of the machine word. We assume the w-bit moduli are multi-precision. Denote s and t the numbers of machine words to represent a w-bit value and a v-bit one, respectively, i.e. $s = \lceil w/m \rceil$, $t = \lceil v/m \rceil$. For a fair comparison, we provide two methods to perform the multi-precision multiplication: the textbook multiplication, and the Karatsuba multiplication [17].

If one employs the textbook multiplication, and takes the number of m-bit multiplications as the complexity metric, the complexity expression of one modulo-P multiplication is:

$$2n^2ts + 5ns^2 + 5ns + 4nt + 9n \tag{9}$$

Notice that $\hat{\lambda} \cdot \tilde{M}_{\mathfrak{B},j}$ or $\hat{\lambda} \cdot \tilde{M}_{\mathfrak{C},j}$ can be carried out by an accumulator or a look-up table since λ is only a few bits. Moreover, the multiplication by d_i in the channel reduction (c.f. Algorithm 1) can be performed by additions if the Hamming weight of d_i is small. If excluding these multiplications, the total number of m-bit multiplications for a modular multiplication is:

$$2n^2ts + 5ns^2 \tag{10}$$

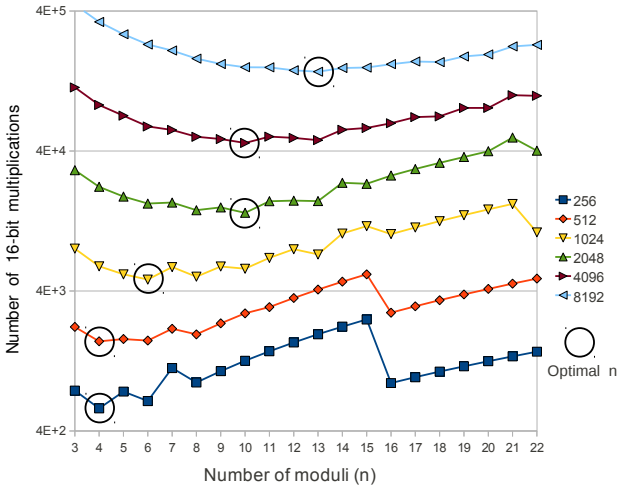


Fig. 2: Number of 16-bit multiplications for one modular multiplication and best choice of n (circled in the figure) for different bit-length of P using textbook multiplication on a 16-bit machine ($m = 16$).

For comparison, the complexity of the digit-serial Montgomery algorithm is: [4]

$$2n^2s^2 + ns \tag{11}$$

Therefore, as long as $t < s - (5s-1)/2n$, the complexity of the refined RNS Montgomery algorithm is less than that of the digit-serial Montgomery algorithm. If lazy reduction is available, the advantage of the refined RNS Montgomery algorithm will be more obvious. Interested readers can also derive the results for cases which use Karatsuba multiplication and/or count the number of additions.

5 RNS Parameter Selection

5.1 Selection of n, the Number of Moduli

The complexity of RNS Montgomery multiplication thus depends on n and v. According to the experiments, we observe that v, and consequently t, are also mainly determined by n. The following second-order polynomial can be used to estimate v and v' (c.f. Section 5.3 and Fig. 3(a)).

$$\hat{v} = 0.1302n^2 + 4.2215n - 7.3003 \tag{12}$$
$$\hat{v'} = 0.1174n^2 + 2.4932n - 4.7379 \tag{13}$$

Therefore, the complexity can be considered as a function with the solely variable n. Given P and m, we can easily enumerate all possible values of n and choose the optimal. Fig. 2 shows the best choices of n for different size of P on a 16-bit machine ($m = 16$). We use the number of 16-bit multiplications as the performance indicator, and it is calculated by (9). For each case, the envelope of the curve is convex, and we can always find an optimal n (circled in the figure).

4. The number of machine words to represent an integer less than P is approximately ns.

TABLE 2: Operations of RNS Montgomery modular multiplication (Algorithm 3)

Steps	Operations	# Operations	# m-bit MUL/operation Plain	# m-bit MUL/operation Karatsuba	# m-bit MUL Plain	# m-bit MUL Karatsuba
Basic Modular Multiplication						
1–4, 7	$w \times w$-bit MUL	$5n$	s^2	$s^{\log_2 3}$	$5ns^2$	$5ns^{\log_2 3}$
6, 9	$v \times w$-bit MUL	$2n^2$	ts	$t^{\log_2 3-1}s$	$2n^2ts$	$2n^2t^{\log_2 3-1}s$
6, 9	$\lceil \log_2 n \rceil \times \lceil \frac{vn}{n-1} \rceil$-bit MUL‡	$2n$	$t+1$	$t+1$	$2nt+2n$	$2nt+2n$
1–4, 7	$2w$-bit Reduction†	$5n$	$s+1$	$s+1$	$5ns+5n$	$5ns+5n$
6, 9	$(w+v+\lceil \log_2 n \rceil)$-bit Reduction†‡	$2n$	$t+1$	$t+1$	$2nt+2n$	$2nt+2n$
7	w-bit Mod ADD/SUB	n	-	-	-	-
5, 8	$(w+\lceil \log_2 n \rceil)$-bit ADD*	$2n-1$	-	-	-	-
6, 9	$(w+v+\lceil \log_2 n \rceil)$-bit ADD	$2n^2$	-	-		-
One More Addition Term for Lazy Reduction						
1, 2	$w \times w$-bit MUL	$2n$	s^2	$s^{\log_2 3}$	$2ns^2$	$2ns^{\log_2 3}$
-	$2w$-bit ADD	$2n$	-	-	-	-

† It is not counted when multiplications by d_i is performed by additions (Algorithm 1), and we assume d_i fits in one word.
‡ $\lceil \log_2 n \rceil$ is the bit-length of λ, and $\lceil vn/(n-1) \rceil$ the bit-length of $\tilde{M}_{\mathfrak{B},j}$ or $M_{\mathfrak{C},j}$.
∗ This is for the approximation of λ, and even smaller bit-length (e.g. 8-bit) will work [9].

Note that $M_{\mathfrak{B}} = \prod_{i=1}^{n}(2^w - d_i) \simeq 2^{nw}$ and $M_{\mathfrak{B}} > 3P$. Therefore, w is approximately $\lceil \lceil \log_2 3P \rceil / n \rceil$. Once n (and thus w) is selected, we can proceed to choose the RNS base moduli. We provide two methods: MPP and FCFS$^+$.

5.2 Multiple-Plus-Prime (MPP)

The following theorem provides a determinative way to construct base $\mathfrak{B}$ and base $\mathfrak{C}$. The proof is not included due to the paper length limitation.

Theorem 1: Let $\mathbb{P} = \{1, 2, 3, 5, 7, 11, 13, \ldots \delta\}$, i.e. a set that includes 1 and the first $(2n-1)$ primes. Let $\Theta := \prod_{i=1}^{2n} p_i, p_i \in \mathbb{P}$, and M a multiple of Θ, then all the $M + p_i, p_i \in \mathbb{P}$, are pairwise coprime.

Given w and n, define $M := 2^w - |2^w|_\Theta$, then M is a multiple of Θ. If $|2^w|_\Theta - p_i < 2^{\lfloor w/2 \rfloor}, \forall p_i \in \mathbb{P}$, then $M + p_i$ are pseudo-Mersenne numbers ($d_i = |2^w|_\Theta - p_i$). Hence, the elements of $\mathfrak{M} = \{M + p_i : p_i \in \mathbb{P}\}$ can then serve as the moduli in $\mathfrak{B}$ and $\mathfrak{C}$. This method is named Multiple-Plus-Prime (MPP) approach.

MPP approach is deterministic and the value of $\tilde{B}_{i,j}$ is determined by n alone. Hence, it takes no intensive computations to generate bases. More importantly, MPP proves that there must exist a set of coprimes $\mathfrak{M}$ consisting of $2n$ coprime moduli and $\max(\mathfrak{M}) - \min(\mathfrak{M}) < \varphi(2n)$, where $\varphi(2n)$ is the $2n$-th prime. However, $d_i = |2^w|_\Theta - p_i$ is not likely to be a small integer or with small Hamming weight when n is large, and consequently, the channel reduction (Algorithm 1) cannot be performed efficiently. Therefore, MPP is only attractive when n is small, and multiplications by d_i are not performed by additions. A more practical method that leads to good RNS bases for larger n is provided in the following section.

5.3 First Come, First Selected Improved (FCFS$^+$)

This method only selects pseudo-Mersenne moduli, thus guarantees an efficient channel reduction.

Algorithm 4 First Come, First Selected Improved (FCFS$^+$) base selection algorithm

Require: w and n
Ensure: $\mathbb{T} = \{T_1, \ldots, T_{2n}\} = \{2^w - d_1, \cdots, 2^w - d_{2n}\}$, where $T_j, 1 \leqslant j \leqslant 2n$ are pairwise co-prime.
1: $blacklist \leftarrow \phi$
2: **repeat**
3: $previous\_blacklist \leftarrow blacklist$
4: $i \leftarrow 1$; $\mathbb{T} \leftarrow \phi$;
5: **repeat**
6: **if** $(2^w - i)$ is co-prime to all elements in $\mathbb{T}$ **and** $(2^w - i) \notin blacklist$ **then**
7: $\mathbb{T} \leftarrow \mathbb{T} \cup \{2^w - i\}$.
8: **end if**
9: $i \leftarrow i + 2$; \\ even integers are excluded.
10: **until** #$\mathbb{T}$=$2n$.
11: **for** $j = 1 \rightarrow 2n$ **do**
12: $f_2 \leftarrow$ the second least prime factor of T_j †
13: **if** $f_2 < (T_j - T_{2n})/2$ **then**
14: $blacklist \leftarrow blacklist \cup \{T_j\}$.
15: **end if**
16: **end for**
17: **until** $previous\_blacklist = blacklist$
18: **return** $\mathbb{T}$
† If T_j has no second least factor, $f_2 \leftarrow \infty$

Steps 4–10 of Algorithm 4 describes one straightforward way to select $2n$ odd comprime moduli by the "First Come, First Selected (FCFS)" principle: we search down from $2^w - 1$ until we get $2n$ coprimes, as the numbers near 2^w are pseudo-Mersenne numbers and often yield low Hamming weight.

The coprimes provided by the FCFS method are usually not very close to each other, as numbers which come first will affect the recruitment of the following numbers. For instance, $2^w - 7$ and $2^w - 11$, where

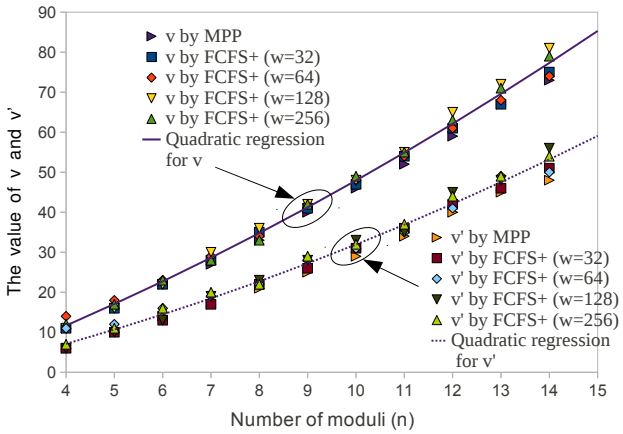
$w = 64$, are coprime, but none of them are coprime with $2^w - 1$. Hence, for $w = 64$, it is better to exclude $2^w - 1$ so that the selected coprimes are closer. We propose an improved version of FCFS (hense FCFS$^+$) to perform the refinement.

Steps 12-15 examine whether T_j is blocking other candidates to be recruited. Let f_1, f_2 be the first and second least distinct prime factor of T_j, respectively (hence, $f_1 < f_2$).[5] Therefore, two odd number, $T_j - 2f_1, T_j - 2f_2$, are not coprime with T_j, and hence excluded as T_j comes first.[6] If $T_j - 2f_2 > T_{2n}$, then $T_{2n} < T_j - 2f_2 < T_j - 2f_1 < T_j \leqslant T_1$. We can then blacklist T_j. In the next round (Steps 2–18), since T_j is blacklisted and not considered anymore, we may include two other numbers $T_j - 2f_1$ and $T_j - 2f_2$. Consequently, the element selected in the next iteration may get closer.

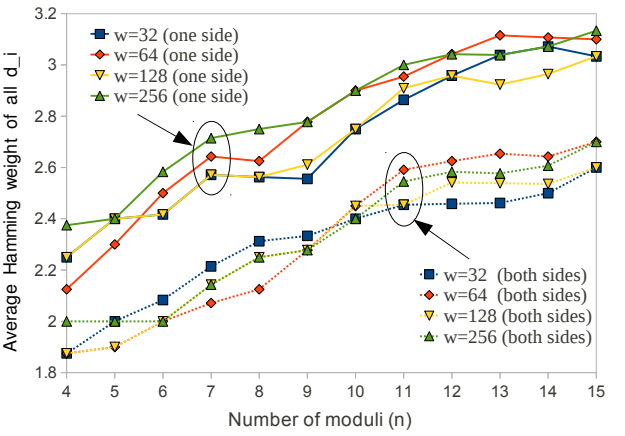
Note that FCFS$^+$ does not guarantee the best base selection, as $T_j - 2f_1$ or/and $T_j - 2f_2$ may still have small factors and go into the blacklist. Nevertheless, FCFS$^+$ generally provides closer moduli than FCFS.

Fig. 3(a) shows the value of v and v' for moduli selected by FCFS$^+$ and MPP for different n and w. According to it, v and v' are generally determined by n, the number of moduli, and has little relation with w. Base on this experiment, we obtain (12) and (13) by the second-order polynomial curve-fitting.

Fig. 3(b) shows the average Hamming weight of all d_i. One can also develop the variants of Algorithm 4, e.g. to recruit the moduli from both sides of 2^w on the number axes (i.e. d_i can be 0, positive or negative). The average Hamming weights of the moduli from both sides of 2^w are also shown in Fig. 3(b) using dashed line. Generally speaking, the Hamming weight of d_i also increases with the growth of n.



(a) The value of v and v'



(b) Average Hamming weight of moduli

Fig. 3: The value of v and v', and average Hamming weight of moduli selected by FCFS$^+$.

5.4 RNS Parameter Selection Process

To summarize, we propose the following process to select parameters:

1) Emulate n from 2 to $\lceil \lceil \log_2 3P \rceil / m \rceil$;
2) For each n value, compute the w, s, v, t , and the complexity index;
3) Select the optimal n and its corresponding w;
4) Select $2n$ coprimes using FCFS$^+$ given n and w.

For $n > \lceil \lceil \log_2 3P \rceil / m \rceil$, w will be less than m, and the platform will not be utilized efficiently. Therefore, for Step 1), we only need to emulate from 2 to $\lceil \lceil \log_2 3P \rceil / m \rceil$. For Step 2), $w = \lceil \lceil \log_2 3P \rceil / n \rceil$, v or v' is calculated by (12) or (13), $s = \lceil w/m \rceil$, $t = \lceil v/m \rceil$, and the complexity index is computed by (9), (10) or according to the user's condition. Step 3) are described in Section 5.1, and 4) is discussed in Section 5.3. We use the following example to demonstrate the process.

5. When T_j has no second least prime factor, $f_2 \leftarrow \infty$, and T_j will not be blacklisted.

6. Note that $T_j - f_1, T_j - f_2$ are even, and hence, are not considered here.

256-bit Modular Multiplications on a 16-bit CPU

We first enumerate the n ranging from 2 to 16. Then we compute all the w, s, v, t values, and the complexity index values by (9) as shown in Fig. 2. The optimal n is 4 with $w = 64$, $s = 4$, $t = 1$.

Using Algorithm 4 (FCFS$^+$), two rounds are performed. The details are shown in Table 3. In the 1st round, $T_1 = 2^w - 1$, which has $f_1 = 3$ and $f_2 = 5$, blocks $T_1 - 2f_1$ $(= 2^w - 7)$ and $T_1 - 2f_2$ $(= 2^w - 11)$. After blacklisting it, $2^w - 7$ and $2^w - 11$ are recruited in the 2nd round, and the moduli get closer. After 2nd round, the blacklist stays the same, and Algorithm 4 returns the moduli. We group the two bases and let $\mathbb{L}_X$ to show the bit-length of all $X_{i,j}$ in matrix format:

$$\mathfrak{B} = \{2^w - 33, 2^w - 15, 2^w - 7, 2^w - 3\},$$
$$\mathfrak{C} = \{2^w - 17, 2^w - 11, 2^w - 9, 2^w - 5\}.$$

$$\mathbb{L}_{\tilde{\mathfrak{B}}} = \begin{pmatrix} 10 & 10 & 7 & 6 \\ 9 & 10 & 9 & 7 \\ 9 & 10 & 10 & 8 \\ 9 & 9 & 12 & 9 \end{pmatrix}, \quad \mathbb{L}_{\tilde{\mathfrak{C}}} = \begin{pmatrix} 10 & 8 & 8 & 7 \\ 7 & 7 & 6 & 5 \\ 6 & 7 & 7 & 8 \\ 14 & 14 & 14 & 14 \end{pmatrix}.$$

TABLE 3: Example of base selection using FCFS$^+$ for $n = 4, w = 64$

		T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
1st round	T_j	2^w-1	2^w-3	2^w-5	2^w-9	2^w-15	2^w-17	2^w-33	2^w-39
	1st factor	3	13	11	7	53	19	827	$>10^4$
	2nd factor	5	3889	59	9241	$>10^4$	67	$>10^4$	$>10^4$
2nd round	T_j	2^w-3	2^w-5	2^w-7	2^w-9	2^w-11	2^w-15	2^w-17	2^w-33
	1st factor	13	11	3	7	5	53	19	827
	2nd factor	3889	59	$>10^4$	9241	2551	$>10^4$	67	$>10^4$

TABLE 4: BE using CRT versus BE using MRS

Transform	Phase I	Phase II
CRT-based	Compute ξ_i n mul. by B_i^{-1}	Compute $\lvert Q\rvert_{c_i}$ n^2 mul. by $\tilde{B'}_{i,j}$
MRS-based	Compute $\bar{q}_i$ $\frac{n(n-1)}{2}$ mul. by $\lvert b_i^{-1}\rvert_{b_j}$	Compute $\lvert Q\rvert_{c_i}$ $n(n-1)$ mul. by $\lvert b_i\rvert_{c_j}$

After zero truncation:

$$\mathbb{L}_{\tilde{\mathfrak{B}}'} = \begin{pmatrix} 6 & 6 & 3 & 3 \\ 4 & 5 & 4 & 4 \\ 4 & 4 & 4 & 1 \\ 3 & 3 & 6 & 6 \end{pmatrix}, \ \mathbb{L}_{\tilde{\mathfrak{C}}'} = \begin{pmatrix} 5 & 3 & 5 & 2 \\ 3 & 3 & 3 & 1 \\ 2 & 3 & 4 & 4 \\ 6 & 7 & 5 & 8 \end{pmatrix}.$$

Now we have:

$$v = \max\{\mathbb{L}_{\tilde{\mathfrak{B}}}, \mathbb{L}_{\tilde{\mathfrak{C}}}\} = 14, v' = \max\{\mathbb{L}_{\tilde{\mathfrak{B}}'}, \mathbb{L}_{\tilde{\mathfrak{C}}'}\} = 8.$$

The size of $\lvert B_i\rvert_{c_j}$ is reduced from 64-bit to 14-bit or 8-bit (from four 16-bit words to one word). Hence, the original 64×64 multiplications can be replaced by 14×64 ones or 8×64 ones.

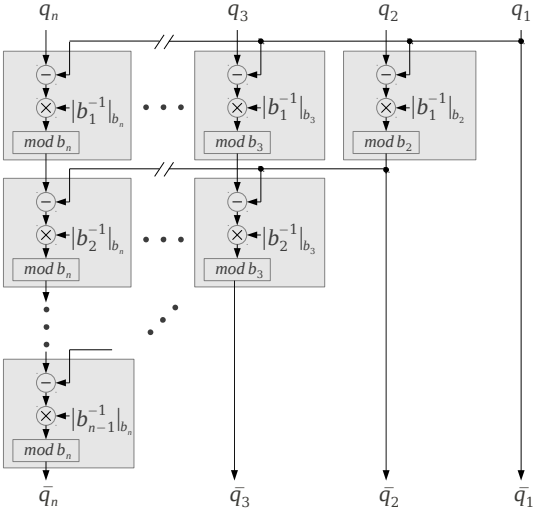


Fig. 4: The data flow for transformation from RNS to MRS.

6 Discussions

6.1 CRT versus MRS

When using MRS-based BE, the computation employs two phases: an RNS-to-MRS transform (Phase I) and an MRS-to-RNS transform (Phase II). For comparison reason, the CRT-based method can also be separated into 2 phases: Phase I to compute ξ_i, and Phase II the matrix-by-vector multiplication. Table 4 summarizes the complexity of MRS-based BE along with the comparison with the CRT-based BE.

Phase I of MRS-base method is to transform the RNS representation of Q, $\{q_n, q_{n-1}, \ldots, q_1\}$, to $\{\bar{q}_n, \bar{q}_{n-1}, \ldots, \bar{q}_1\}$, where $\bar{q}_i < b_i$ and satisfies the following equation:

$$Q = \Big[\ldots\big[[\bar{q}_n]\cdot b_{n-1} + \bar{q}_{n-1}\big]\cdot b_{n-2} + \cdots + \bar{q}_2\Big]\cdot b_1 + \bar{q}_1 \quad (14)$$

Fig. 4 shows the data flow of Phase I. Each element in Fig. 4 performs the patten $\bar{x}_j \leftarrow \big|(\bar{x}_j - \bar{q}_i)\cdot|b_i^{-1}|_{b_j}\big|_{b_j}$. Bajard, Meloni and Plantard showed the multiplication by $|b_i^{-1}|_{b_j}$ can be accelerated when b_i and b_j are *close* [18]. Note that $|b_i^{-1}|_{b_j} = |(b_i - b_j)^{-1}|_{b_j}$. The main idea is to use the Montgomery reduction, which computes $|xR^{-1}|_p$, to perform $\big|(\bar{x}_j - \bar{q}_i)\cdot|b_i^{-1}|_{b_j}\big|_{b_j}$. The division by $(b_i - b_j)$, which is required in the Montgomery reduction, is performed with Barrett's algorithm [19]. However, the actual efficiency of this algorithm in hardware or software is not clear. The relatively complex structure of the Montgomery reduction algorithm may lead to more clock cycles, even if the size of the operands are small.

Bajard, Kaihara and Plantrad showed how to select moduli such that $|b_i^{-1}|_{b_j}$ has low Hamming weight in Non-Adjacent Form (NAF) [20]. They suggested to choose moduli that have the form 2^k, $2^k - 1$ and $2^k - 2^t \pm 1$. However, the Hamming weight of $|b_i^{-1}|_{c_j}$ increases quickly with the growth of n. Using the suggested parameters by [20], the maximal Hamming weight is 5 for $n = 4$, while it is 17 for $n = 6$. Note that for FCFS$^+$ method, the maximal Hamming weight of $\tilde{B}'_{i,j}$ is 4 and 6 for $n = 4$ and $n = 6$, respectively.

Phase I of MRS-based BE, as shown in Fig. 4, has a triangle data-flow. As a result, it is difficult to parallelize. While for CRT-based BE, the computation of $\xi_{Q,i}$ for $1 \leqslant i \leqslant n$ can be performed in parallel. Note that one of the main reasons for using RNS is that it's easy to parallelize.

Phase II is to compute $\{|Q|_{c_n}, |Q|_{c_{n-1}}, \ldots, |Q|_{c_1}\}$ from the MRS representation of Q. Note that $|Q|_{c_i}$ can be computed by involving $(n-1)$ times the following operation: $x'_i \leftarrow |x'_i \cdot b_j + \bar{q}_j|_{c_i}$. It can also be computed as an inter product of two vectors:

$$|Q|_{c_i} = \left|[\bar{q}_n, \bar{q}_{n-1}, \ldots, \bar{q}_1]\cdot\Big[\Big|\prod_{k=1}^{n-1} b_k\Big|_{c_i}, \Big|\prod_{k=1}^{n-2} b_k\Big|_{c_i}, \ldots, 1\Big]^T\right|_{c_i} \quad (15)$$

Note that the close moduli can also reduce the bitlength of of $|b_j|_{c_i}$ or $\left|\prod_{k=1}^{n-j} b_k\right|_{c_i}$ in (15).

6.2 Implementations using Proposed Method

We used the proposed method to select parameters for pairing implementations on FPGA [12], [16]. The Cox-Rower architecture have been used in these works, and a 256-bit modulo operation is reduced from 19 cycles to 5 cycles on the similar hardware architecture.

There are two designs in [12] and they both choose $n = 8$. Design I uses the full-length multiplication for BE, and it needs 19 cycles for one modular reduction. Design II uses $\tilde{B}'_{i,j}$ (25-bit) to substitute $|B_i|_{c_j}$ (34-bit), and a modular reduction only takes 12 cycles. However, the optimal parameter is $n = 4$ for $\lceil \log_2 P \rceil = 256$ and $m = 18$ (the FPGA has pre-mounted 18-bit multipliers). Therefore, our following work [16] chooses $n = 4$ and $w = 67$, and one modular reduction only takes 6 cycles. With precomputations as Algorithm 3, it can be further reduced to 5 cycles on the same architecture as [16].

The implementation details are out of the scope of this paper. Nonetheless, the acceleration of pairing computations in [12], [16] has demonstrated the advantage of the refined parameter selection methods.

7 Conclusions and Future Work

In this paper, we observe that when moduli are close to each other, the complexity of base extensions using CRT can be reduced, and consequently, RNS Montgomery algorithm is accelerated. Based on the analysis of the refined RNS algorithm, we propose a systematic parameter selection procedure, which emphasizes on the selection of the optimal n, the number of moduli. Subsequently, MPP and FCFS$^+$ methods are proposed to select the bases consisting of $2n$ close moduli. Our complexity analysis and the experimental results prove the advantage of the refined RNS base selection methods.

For the future work, we will investigate theoretical lower bound of the complexity. One possible way to find the optimal bases is considering the base selection as a weighted clique problem [21]. All pseudo-Mersenne numbers in range $[2^w - 2^{\lfloor w/2 \rfloor}, 2^w + 2^{\lfloor w/2 \rfloor}]$ are the vertices, and two vertices, say x_i and x_j, are adjacent if they are coprime. The problem is thus to find a fixed-size clique with the minimum $\prod_{i \neq j} |x_i - x_j|$. We believe that MPP method provides a hint (or even a benchmark) on the distribution of the coprime moduli.

Acknowledgments

The authors would like to thank the anonymous viewers for the constructive comments. We also would like to thank Frederik Vercauteren, Nicolas Guillermin, and Hongbing Fan for the helpful discussions. This work is partly supported by the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No. CityU 123612) and CityU Strategic Research Grant (Project No. 7008185), and partly by the Flemish Government through FWO G.0550.12N and G.0130.13, and the Hercules Foundation AKUL/11/19.

References

[1] W. Diffie and M. Hellman, "New directions in cryptography," *Information Theory, IEEE Transactions on*, vol. 22, no. 6, pp. 644–654, 1976.

[2] V. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology – CRYPTO 1985*, ser. LNCS. Springer, 1986, vol. 218, pp. 417–426.

[3] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987.

[4] N. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," in *Public Key Cryptography – PKC 2010*, ser. LNCS. Springer, 2010, vol. 6056, pp. 420–443.

[5] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to Parallel Computing (2nd Edition)*. Addison Wesley, 2003.

[6] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[7] K. Posch and R. Posch, "Modulo reduction in residue number systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 6, no. 5, pp. 449–454, 1995.

[8] J. Bajard, L. Didier, and P. Kornerup, "An RNS Montgomery modular multiplication algorithm," *Computers, IEEE Transactions on*, vol. 47, no. 7, pp. 766–776, 1998.

[9] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-rower architecture for fast parallel Montgomery multiplication," in *Advances in Cryptology – EUROCRYPT 2000*, ser. LNCS. Springer, 2000, vol. 1807, pp. 523–538.

[10] H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura, "Implementation of RSA algorithm based on RNS Montgomery multiplication," in *Cryptographic Hardware and Embedded Systems – CHES 2001*, ser. LNCS. Springer, 2001, vol. 2162, pp. 364–376.

[11] N. Guillermin, "A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$," in *Cryptographic Hardware and Embedded Systems – CHES 2010*, ser. LNCS. Springer, 2010, vol. 6225, pp. 48–64.

[12] R. Cheung, S. Duquesne, J. Fan, N. Guillermin, I. Verbauwhede, and G. Yao, "FPGA implementation of pairings using residue number system and lazy reduction," in *Cryptographic Hardware and Embedded Systems – CHES 2011*, ser. LNCS. Springer, 2011, vol. 6917, pp. 421–441.

[13] Ç. K. Koç, T. Acar, and B. S. Kaliski, "Analyzing and comparing Montgomery multiplication algorithms," *Micro, IEEE*, vol. 16, no. 3, pp. 26–33, 1996.

[14] F. Gandino, F. Lamberti, P. Montuschi, and J. Bajard, "A general approach for improving RNS Montgomery exponentiation using pre-processing," in *20th IEEE Symposium on Computer Arithmetic – ARITH 2011*, 2011, pp. 195 –204.

[15] K. Posch and R. Posch, "Base extension using a convolution sum in residue number systems," *Computing*, vol. 50, pp. 93–104, 1993.

[16] G. Yao, J. Fan, R. Cheung, and I. Verbauwhede, "Faster pairing coprocessor architecture," in *Pairing-Based Cryptography – Pairing 2012*, ser. LNCS. Springer, 2012, p. To appear.

[17] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Soviet Physics Doklady*, vol. 7, no. 7, pp. 595–596, 1963.

[18] J. Bajard, N. Meloni, and T. Plantard, "Efficient RNS bases for cryptography," in *World Congress: Scientific Computation Applied Mathematics and Simulation – IMACS 2005*, no. 1, 2005, pp. 1–7.

[19] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Advances in Cryptology – CRYPTO 86*, ser. LNCS. Springer, 1987, vol. 263, pp. 311–323.

[20] J. Bajard, M. Kaihara, and T. Plantard, "Selected RNS bases for modular multiplication," in *19th IEEE Symposium on Computer Arithmetic – ARITH 2009*, 2009, pp. 25–32.
[21] M. Golumbic, *Algorithmic graph theory and perfect graphs*. North-Holland, 2004, vol. 57.